Project Title: Smart Homes

Case Study: Forecasting Energy Consumption for Improved Efficiency in Smart Homes

Description:
This project aims to predict energy consumption in smart homes by analyzing appliance activity, room occupancy, and external factors such as weather. The goal is to identify key factors influencing energy use and provide insights to optimize energy efficiency and reduce environmental impact.

Problem Statement:
Human activities are a major contributor to climate change, significantly impacting nature, other species, and future generations. To create a sustainable future, we need to adopt more mindful habits and work together. Improving energy efficiency in our homes is a crucial step in this direction.

Dataset: This project utilized a dataset downloaded from kaggle (link here: https://www.kaggle.com/datasets/taranvee/smart-home-dataset-with-weather-information). The dataset contains over 500,000 data points recorded at 1-minute intervals and covers entire year of 2016. There are 32 predictors including details on how much energy each room and appliance used every minute, along with the weather outside and how much energy the house itself produced.

Tools and Technologies Used:
The project was implemented in **Python**, utilizing the following libraries and tools:

- **Matplotlib** and **Seaborn**: For data visualization
- **NumPy** and **Pandas**: For data manipulation and analysis
- **Scikit-learn**: For machine learning tasks, such as accuracy analysis, creating test/train datasets
- **TensorFlow**: For deep learning predictive modeling
- **Statsmodels**: For time series modelling

Challenges:
The dataset contains 32 predictors that exhibit correlations with one another. It also includes challenges such as missing values, mixed data types (e.g., both string and numerical data within a single column), and complex column names. Prior to modeling, the dataset was thoroughly preprocessed to address these issues.

Results:
While time series analysis is a common approach for this type of dataset, it struggled to capture the significant spike in energy consumption during the summer months.The **best-performing time series model** achieved a **prediction error of 24%** in this project.
In contrast, a simple **feedforward neural network** successfully identified energy consumption patterns and achieved a **prediction error** as low as **13%**, demonstrating its effectiveness in modeling the data.

1. Motivation of The Study

Technology enables us to monitor both energy consumption and production in smart homes. By incorporating solar panels, households can generate their own energy, taking a step toward self-sufficiency. Achieving a balance between energy usage and production is essential for building a sustainable future. Reducing our environmental footprint starts with developing mindful energy habits. While this may seem challenging, optimizing energy use in our homes can make a significant difference toward sustainability.

Smart homes generate valuable data, such as energy usage by specific appliances and individual rooms. A publicly available dataset on Kaggle (link) offers comprehensive information, including household energy consumption, weather conditions, and energy generation. This data serves as a crucial resource for analyzing and optimizing energy usage patterns.

Benefits of Prediction of Energy Consumption in a Smart Home:

1. Identifying energy-heavy appliances allows you to replace them with more efficient models and adopt smarter usage habits.

2. Reducing energy consumption not only lowers your carbon footprint but also lowers your energy bills.

3. Making informed decisions about energy-saving strategies, such as understanding peak electricity rates (which are typically higher between 4 PM and 9 PM), and optimizing appliance use during periods of peak solar energy generation (usually around noon) can lead to significant savings.

2. Data Cleaning, Preprocessing

2.1. Import Libraries

```
# import libraries here
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

import tensorflow as tf
from tensorflow.keras import layers, models

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error

import pmdarima as pm
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.graphics.gofplots import qqplot
from pmdarima import auto_arima
import warnings
warnings.filterwarnings('ignore')
```

2.2. Read Data

```
df = pd.read_csv('./data.csv', low_memory=False)
```

2.3. Explore the Data

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 503911 entries, 0 to 503910
Data columns (total 32 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   time                 503911 non-null  object 
 1   use [kW]             503910 non-null  float64
 2   gen [kW]             503910 non-null  float64
 3   House overall [kW]   503910 non-null  float64
 4   Dishwasher [kW]      503910 non-null  float64
 5   Furnace 1 [kW]       503910 non-null  float64
 6   Furnace 2 [kW]       503910 non-null  float64
 7   Home office [kW]     503910 non-null  float64
 8   Fridge [kW]          503910 non-null  float64
 9   Wine cellar [kW]     503910 non-null  float64
 10  Garage door [kW]     503910 non-null  float64
 11  Kitchen 12 [kW]      503910 non-null  float64
 12  Kitchen 14 [kW]      503910 non-null  float64
 13  Kitchen 38 [kW]      503910 non-null  float64
 14  Barn [kW]            503910 non-null  float64
 15  Well [kW]            503910 non-null  float64
 16  Microwave [kW]       503910 non-null  float64
 17  Living room [kW]     503910 non-null  float64
 18  Solar [kW]           503910 non-null  float64
 19  temperature          503910 non-null  float64
 20  icon                 503910 non-null  object 
 21  humidity             503910 non-null  float64
 22  visibility           503910 non-null  float64
 23  summary              503910 non-null  object 
 24  apparentTemperature  503910 non-null  float64
 25  pressure             503910 non-null  float64
 26  windSpeed            503910 non-null  float64
 27  cloudCover           503910 non-null  object 
 28  windBearing          503910 non-null  float64
 29  precipIntensity      503910 non-null  float64
 30  dewPoint             503910 non-null  float64
 31  precipProbability    503910 non-null  float64
dtypes: float64(28), object(4)
memory usage: 123.0+ MB
```

2.4. Parse and correct the time column

```
df['time'] = pd.to_datetime(pd.to_numeric(df['time'], errors='coerce'), unit='s')
df[['time']].head()
```

	time
0	2016-01-01 05:00:00
1	2016-01-01 05:00:01
2	2016-01-01 05:00:02
3	2016-01-01 05:00:03
4	2016-01-01 05:00:04

```
# Generate a DatetimeIndex with minute increments
df['time'] = pd.date_range('2016-01-01 05:00', periods=len(df), freq='min')
df[['time']].head()
```

2.5. Extract day, week, month features from time column

```
df['year'] = df['time'].dt.year
df['month'] = df['time'].dt.month
df['day'] = df['time'].dt.day
df['weekday'] = df['time'].dt.day_name()
df['weekofyear'] = df['time'].dt.isocalendar().week
df['hour'] = df['time'].dt.hour
df['minute'] = df['time'].dt.minute
```

2.6. Drop Null Values

```
df.dropna(inplace=True)
df.info()
```

2.7. Combine Related Features

a. Combine Furnace Use

```
# Combine Features: Total Energy Usage by Both Furnaces
df['furnace_use'] = df[['Furnace 1 [kW]','Furnace 2 [kW]']].sum(axis=1)
```

b. Combine Kitchen Use

```
# Calculate the Average Usage Across All Kitchens
# Combining 'Kitchen 12', 'Kitchen 14', and 'Kitchen 38' into a single 'Kitchen' column
df['kitchen_use'] = df[['Kitchen 12 [kW]', 'Kitchen 14 [kW]', 'Kitchen 38 [kW]']].mean(axis=1)
```

2.8. Drop Unnecessary Features

```
df.drop(['icon', 'summary', 'cloudCover'], axis=1, inplace=True)
```

```
df.drop(['Kitchen 12 [kW]',  'Kitchen 14 [kW]',  'Kitchen 38 [kW]', 'Furnace 1 [kW]', 'Furnace 2 [kW]'], axis=1,
```

2.9. Change Variable Names for a more clear representation

```
df = df.rename(columns={
    'use [kW]': 'use',
    'gen [kW]': 'gen',
    'House overall [kW]':'house_overall_use',
    'Dishwasher [kW]': 'dishwasher_use',
    'Home office [kW]': 'home_office_use',
    'Fridge [kW]': 'fridge_use',
    'Wine cellar [kW]':'wine_cellar_use',
    'Garage door [kW]': 'garage_door_use',
    'Barn [kW]': 'barn_use',
    'Well [kW]': 'well_use',
    'Microwave [kW]':'microwave_use',
    'Living room [kW]':'living_room_use',
    'Solar [kW]':'solar_use'
})
```

2.10 Remove Correlated Features

```
def find_correlated_features(df, threshold=0.75):
    # Calculate the correlation matrix
    corr_matrix = df.corr().abs()

    # Create a mask to select upper triangle of the correlation matrix
    upper_tri = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

    # Find columns with correlations above the threshold
    correlated_features=[]
    for column in df.columns:
        if any(upper_tri[column] > threshold):
            correlated_features.append(column)

    return  correlated_features
```

```
correlated_features = find_correlated_features(df)
print(f"Number of correlated features in train data: {len(correlated_features)}")
print(f"Correlated features: {correlated_features}")
```

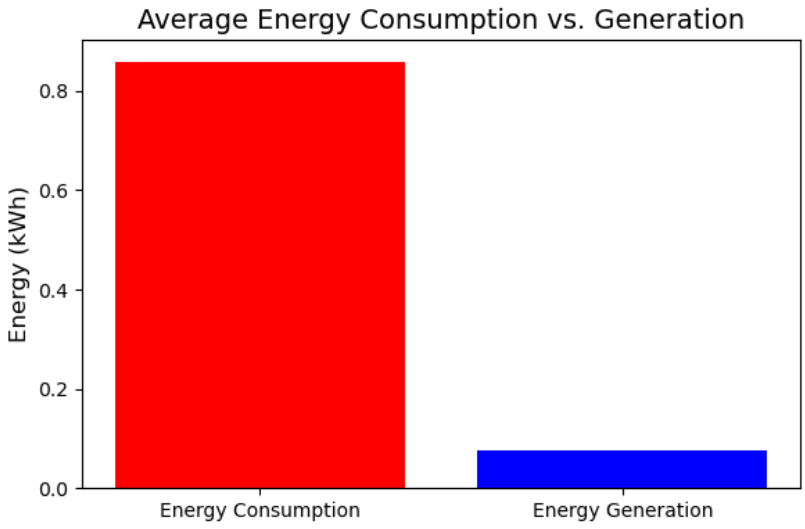
```
Number of correlated features in train data: 6
Correlated features: ['house_overall_use', 'solar_use', 'apparentTemperature', 'dewPoint', 'precipProbability', 'week
ofyear']
```

```
print(f"\n.. Dropping  {len(correlated_features)} correlated variables from the data\n")
df = df.drop(correlated_features, axis=1)
print(f"Shape of the data after removing correlation: {df.shape}")
```

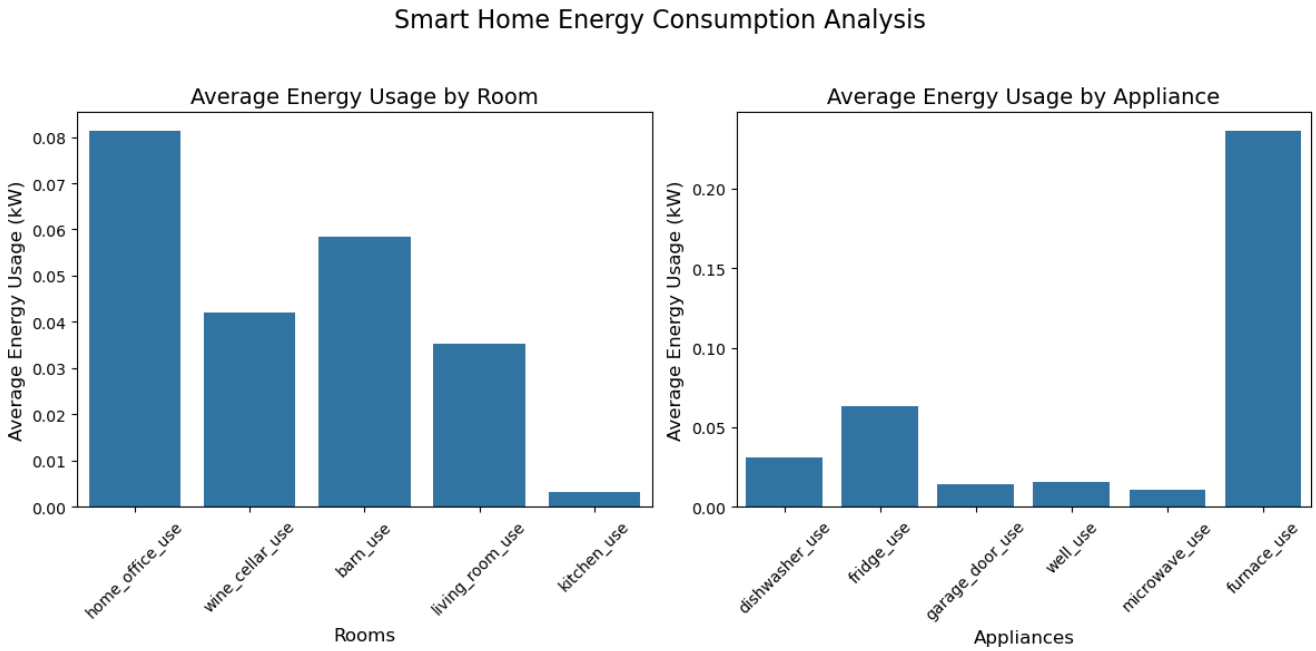
```
.. Dropping  6 correlated variables from the data

Shape of the data after removing correlation: (503910, 25)
```

3. Analyze Patterns Between Energy Consumption, Generation and Weather

3.1 Is there a balance between energy consumption and production?



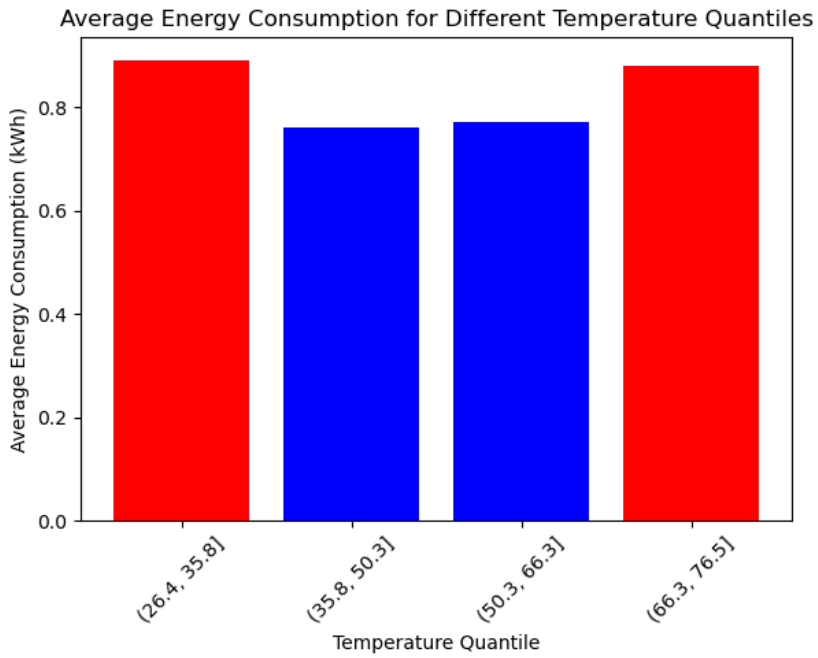
In this smart home, energy consumption is around 5 times higher than energy generation.



3.2. Which appliances and rooms use the most energy?

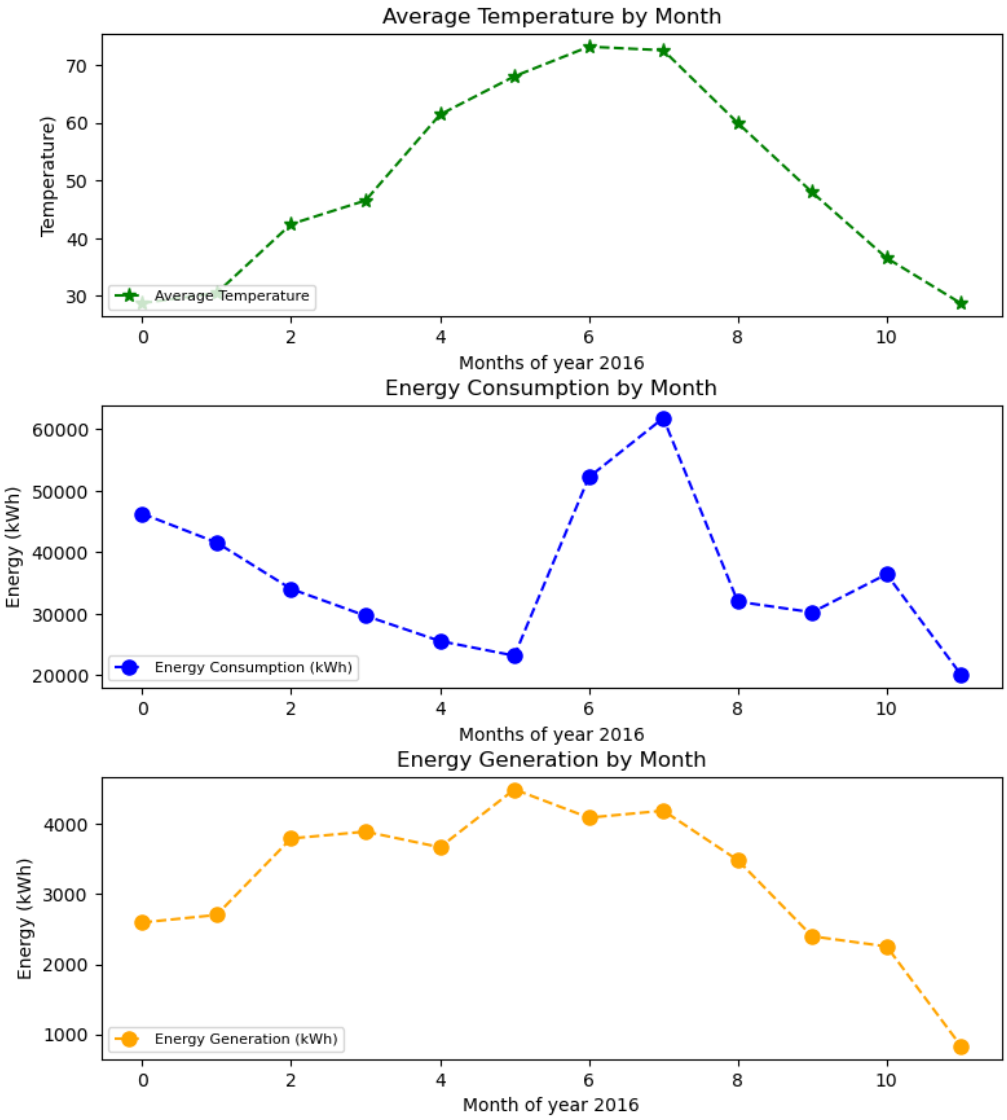
The furnace is the highest energy consumer among the appliances, using approximately 0.23 kW per hour. The home office is the most energy-intensive room, consuming around 0.08 kW per hour, with the fridge usage nearly identical to that of the home office.

3.3. Relationship Between Energy Consumption and Temperature



Energy consumption tends to rise when the temperature drops below 35.8°F or exceeds 66.3°F. This indicates that both extreme cold and hot temperatures contribute to higher energy usage.

3.4 Total Energy Generation and Consumption by Month



Energy Generation and Consumption Peaks:
Both energy generation and consumption reach their highest levels in July and August, aligning with peak summer temperatures.

Energy Consumption Trends:

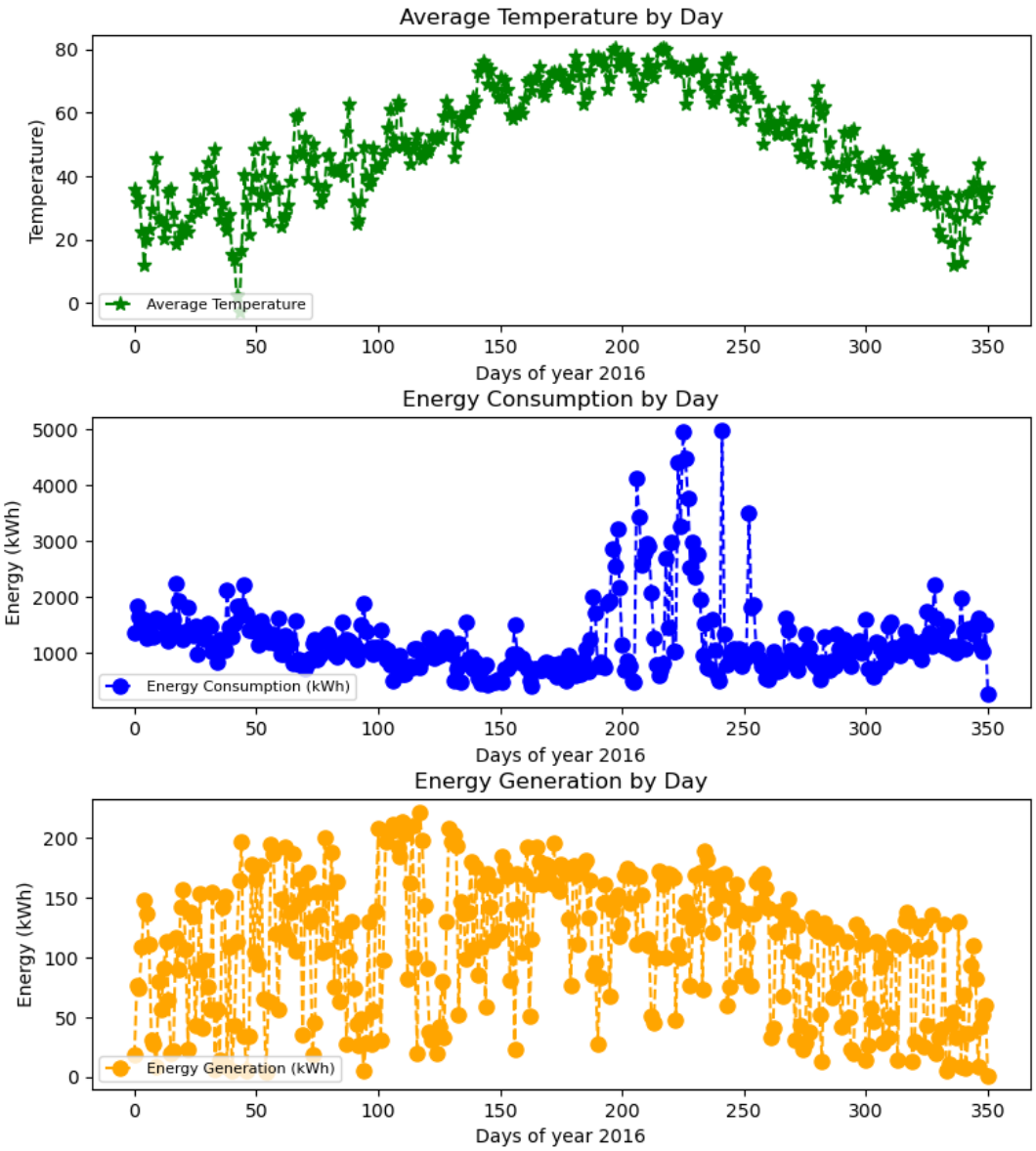
- Energy consumption gradually decreases during the first half of the year.
- A sharp increase occurs in July, peaking in August, likely due to higher air conditioning use during the hottest month.

- After August, consumption drops as temperatures begin to cool.
- Although December is as cold as January, energy consumption in December is notably lower, presenting an interesting anomaly.

Energy Generation Patterns:

- Energy generation is somewhat higher during the summer months compared to the rest of the year.
- However, the generated energy falls significantly short of the energy consumed, revealing a gap between supply and demand.

3.5 Total Energy Generation and Consumption by Day



The visualizations above offer a detailed view of seasonal patterns.

- Temperature steadily rises leading into the summer months, peaks, and then gradually declines.
- Energy consumption remains relatively stable throughout the year, with the exception of the summer, when it experiences a sharp increase.
- Energy generation fluctuates between 0 and 200 kWh per day, while energy consumption varies between 1,000 and 5,000 kWh per day.

4. Modelling

Since time series modeling is commonly used for this type of data, I aim to compare its performance with deep learning methods. To do so, I implemented three different time series models to optimize their parameters, alongside a feedforward neural network. In this section, I will compare the top-performing time series model with the neural network model. For detailed information on all the time series models, please refer to the code notebook.

4.1 Time Series Modelling

Exploratory data analysis revealed a monthly cyclic pattern in energy consumption, with usage typically increasing mid-month, except during the summer when it peaks.

Based on these findings, I will aggregate the data on a daily basis to better analyze the ~30-day (monthly) cyclic trends.

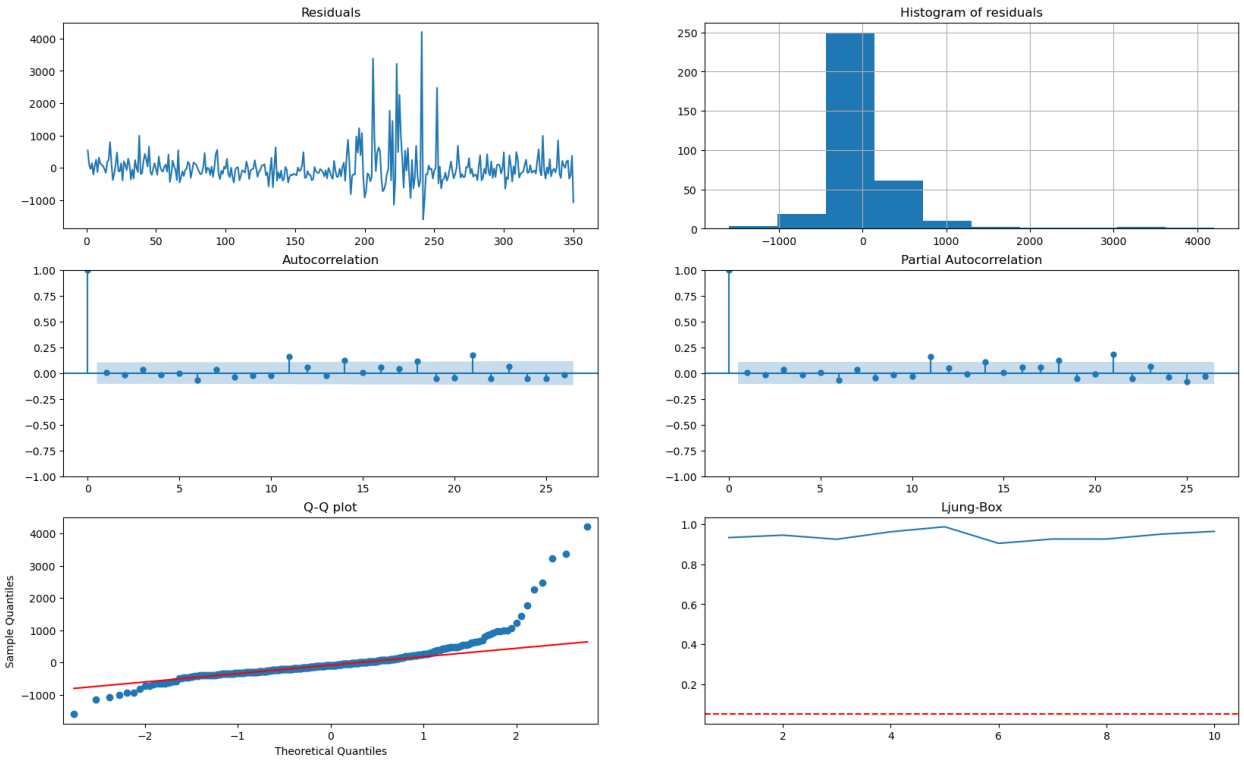
a. Develop a visualize diagnostics functions to evaluate time series modelling

```
# Create a group of plots for diagnostics
def visualize_diagnostics(data):
    # data: residuals of the fitted model
    fig = plt.figure(figsize=(20,12))
    layout = (3, 2)

    data_ax = plt.subplot2grid(layout, (0, 0), colspan=1)
    hist_ax = plt.subplot2grid(layout, (0, 1), colspan=1)
    acf_ax = plt.subplot2grid(layout, (1, 0), colspan=1)
    pacf_ax = plt.subplot2grid(layout, (1, 1), colspan=1)
    qq_ax = plt.subplot2grid(layout, (2, 0), colspan=1)
    lbox_ax = plt.subplot2grid(layout, (2, 1), colspan=1)

    lbox_df = acorr_ljungbox(data, lags=None, return_df=True)

    data.plot(ax=data_ax, title="Residuals")

    data.hist(ax=hist_ax)
    hist_ax.set_title("Histogram of residuals")

    plot_acf(data, ax=acf_ax)
    plot_pacf(data, ax=pacf_ax)

    qqplot(data, ax=qq_ax, line='q')
    qq_ax.set_title("Q-Q plot")

    lbox_ax.plot(lbox_df.lb_pvalue)
    lbox_ax.axhline(y=0.05, color='r', linestyle='--', label='Significance Level (0.05)')
    lbox_ax.set_title("Ljung-Box")
```

b. Apply an ARIMA model

```
# parameters in ARIMA model
p = 1 #Number of lagged observations in the AR component (check PACF plot).

d = 0 #Number of differencing operations to make the series stationary (The data is stationary=>check adfuller test)

q = 1 # Number of lagged forecast errors in the MA component (check ACF plot)


arima_model = ARIMA(df_time_series['use'], order=(p,d,q))
arima_model_fitted = arima_model.fit()

# Print model summary
print(arima_model_fitted.summary())

# Plot diagnostics
visualize_diagnostics(arima_model_fitted .resid[1:])


predictions = arima_model_fitted.predict(start=0, end=len(df_time_series)-1, typ='levels')

# Step 4: Plot actual vs predicted values
plt.figure(figsize=(12, 6))
plt.plot(df_time_series['use'], label="Actual Data", color='blue')
plt.plot(predictions, label="Predicted Data", color='red', linestyle='-')
plt.title("ARIMA Model: Actual vs Predicted Energy Consumption")
plt.xlabel("Time")
plt.ylabel("Energy Consumption")
plt.legend()
plt.tight_layout()
plt.show()
```

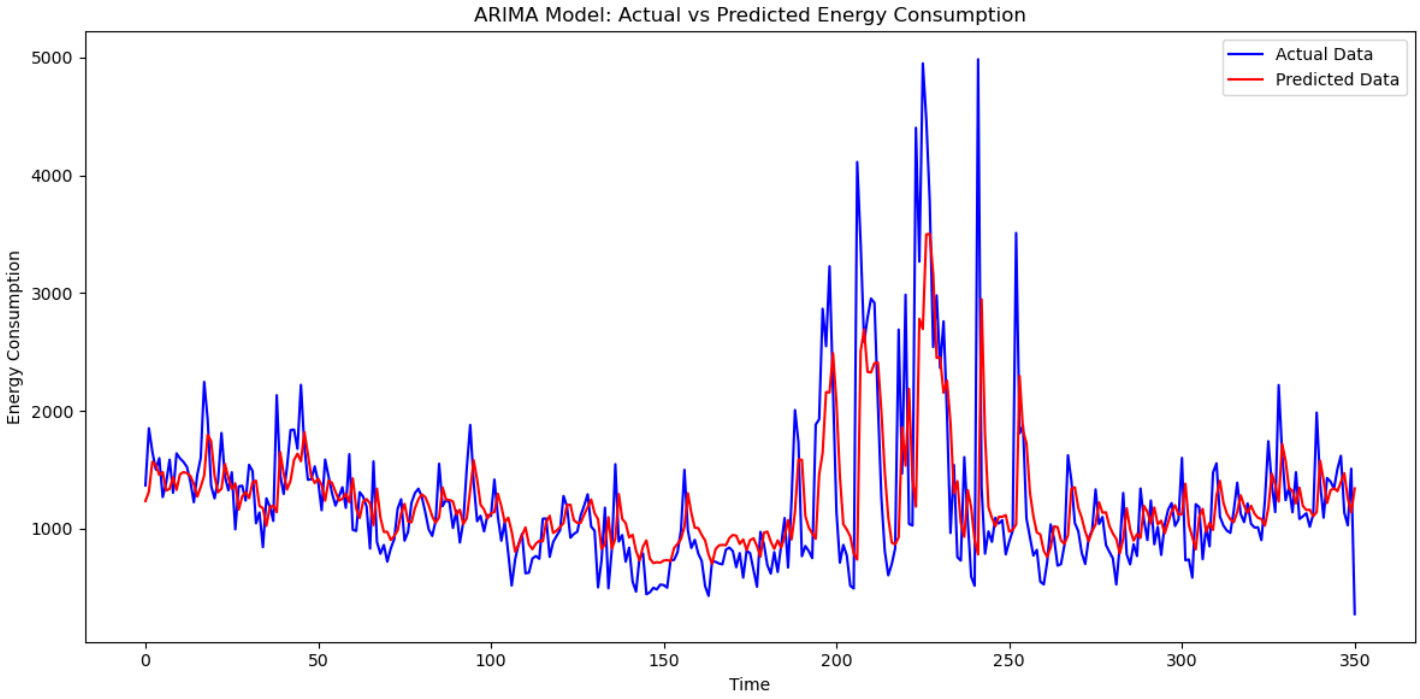
Diagnostic Results:



Residual Analysis:

- The Q-Q plot reveals that the residuals deviate from a normal distribution, suggesting the presence of some remaining patterns or discrepancies.
- However, the Ljung-Box test confirms that the residuals exhibit characteristics of random noise, as expected.

Model Analysis:



While the model successfully captures the main patterns in the data, further refinement may be necessary to address the non-normality detected in the residuals

Mean Absolute Error:
Averge daily energy consumption is 1233 kWh. Arima model can predict daily energy consumption with mean aboslue error of 308 kWh. So prediction error percentage is 25%.

4.2 Feed Forward Neural Network Modelling

Unlike time series modeling, which relies solely on energy consumption data, neural network-based models can incorporate additional predictors. To meet the data volume requirements of neural networks, the model was developed using data aggregated on an hourly basis.

Neural networks tend to perform better when the predictors are standardized.

In the code snippet below, the data is standardized using Scikit-learn's **StandardScaler**. After standardization, the dataset is split into training and testing sets.

a. Standardization and Train-Test Split

```
predictors = ['hour', 'dishwasher_use','home_office_use', 'fridge_use', 'wine_cellar_use', 'garage_door_use',
       'barn_use', 'well_use', 'microwave_use', 'living_room_use',
       'furnace_use', 'kitchen_use', 'temperature', 'humidity', 'visibility',
       'pressure', 'windSpeed', 'windBearing', 'precipIntensity']
target = ['use']
```

```
# Normalize/standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_aggregated_by_hour[predictors])
y = df_aggregated_by_hour[target].values

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y , test_size=0.2, random_state=42)
```

b. Model

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y , test_size=0.2, random_state=42)
```

```
model = models.Sequential()

# Input layer
model.add(layers.InputLayer(input_shape=(X_train.shape[1],)))

# Hidden layers
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(16, activation='relu'))
# Output layer
model.add(layers.Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mae')
```

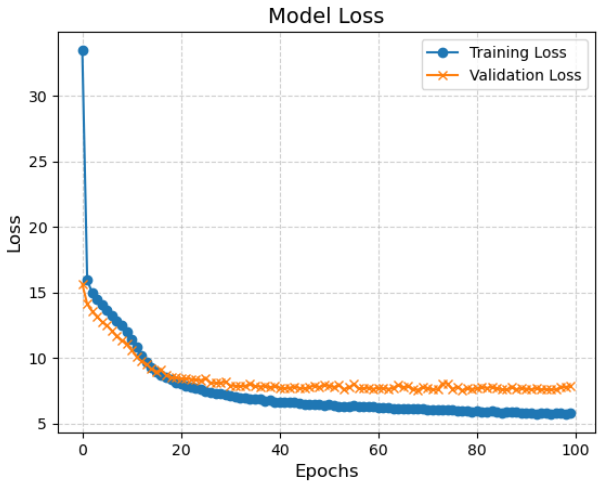
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	1,280
dense_1 (Dense)	(None, 32)	2,080
dense_2 (Dense)	(None, 16)	528
dense_3 (Dense)	(None, 1)	17

Total params: 3,905 (15.25 KB)
Trainable params: 3,905 (15.25 KB)
Non-trainable params: 0 (0.00 B)

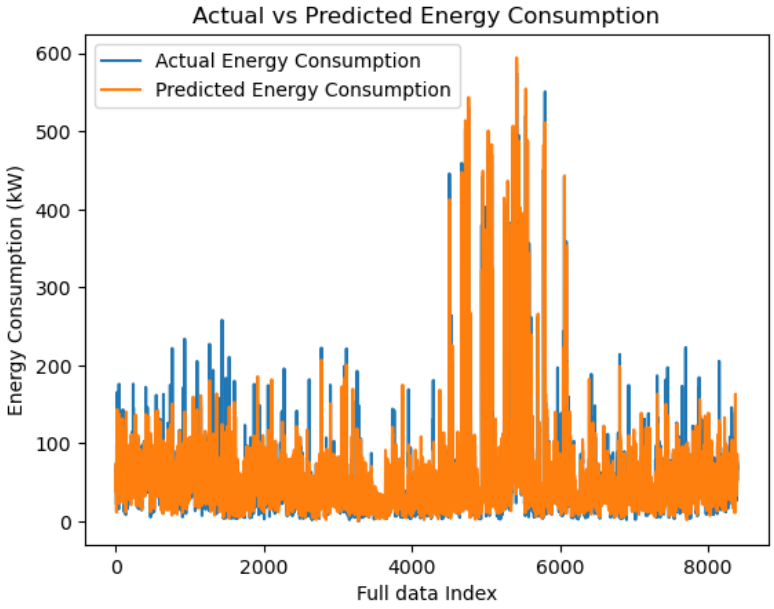
```
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=1)
```



c. Model Performance

The plot above illustrates the training and validation loss as the number of epochs increases. The training mean absolute error (MAE) starts at approximately 34 and decreases to around 6, while the validation loss stabilizes at approximately 7. This represents a significant improvement compared to the performance of the time series models.

c. Does Feed forward Neural Network captures the patterns in the data?



The plot above demonstrates that our model effectively captures the underlying patterns in the data, accurately reflecting the peak energy consumption observed during the summer months.

5. Conclusion

Exploratory data analysis revealed that energy usage peaks during the summer months (July and August) and is closely correlated with temperature. However, diagnostics performed on the time series model indicated that the residuals are not normally distributed, suggesting that the time series approach, with the parameters used in this project, is not well-suited for modeling this dataset. Additionally, the time series model resulted in a prediction error with a mean absolute error (MAE) of approximately 25%.

In contrast, a simple feedforward neural network with three hidden layers and a ReLU activation function effectively captured the peak energy consumption patterns in the data. This model achieved an MAE of 7 for hourly energy consumption predictions, corresponding to approximately 13% prediction error, demonstrating significantly improved performance.